

Some Numerical Experiments in Counting Real Roots of Fewnomials

Walter Moreira

July 28, 2004

1 Introduction

This work is a report on some numerical experiments on counting the number of roots of a specific kind of system of polynomials. It has no definitive results but a survey of attempts, some of them yielding no good results and others being just an idea.

This report has two different parts. The first one describes the problem and some numerical results. The second part of the work is the description of a fast implementation of an algorithm for computing the winding number. This provided a robust and rigorous application for the experiments. The implementation uses free libraries and its free for use and modification. Documentation is provided in the last section.

1.1 Statement of the problem

The problem consists in obtaining bounds for the number of positive real roots of a system of polynomials. But instead of bounding them in function of the degree of the polynomials, we want to compute bounds in function of the number of monomials of the polynomials. This theory began with the work of Khovanski (*Fewnomials*, [Kho91]).

We focus here on a very particular case of this theory. The main reference for most of the notations and results of this part is from [LRW03]. Let's define what we understand by *fewnomials*:

Definition 1. Given polynomials $f_1, \dots, f_k \in \mathbb{R}[X_1, \dots, X_n]$, where f_i has m_i monomials, we say that $F = (f_1, \dots, f_k)$ is a $k \times n$ *fewnomial system* of type (m_1, \dots, m_k) . It is useful to consider here an extension of the polynomials, allowing real exponents.

Consider, then, that F is $n \times n$ fewnomial of type (m_1, \dots, m_n) . We say that a real root ζ of F is *isolated*, if there is no a curve of roots containing ζ (except the point ζ). We also say that ζ is *non-degenerate* if the derivative of F at ζ has full rank. We denote by $\mathcal{N}(m_1, \dots, m_k)$ and $\mathcal{N}'(m_1, \dots, m_k)$ the *maximum* number of positive real isolated and non-degenerate roots of the

fewnomial F , respectively. An standard argument of differential geometry shows that $\mathcal{N}'(m_1, \dots, m_n) \leq \mathcal{N}(m_1, \dots, m_n)$.

Observe that once the positive real roots of a polynomial system are known, that is, the roots in the orthant \mathbb{R}_+^n , we can know the roots on the other orthants by changing x_i by $\pm x_i$. And for knowing the roots on the axis hyperplanes we can set some x_i to zero, reducing the problem to a fewnomial system of “smaller” type.

We concentrate, then, in the case of 2×2 fewnomial systems of type $(3, m)$, for $m = 3, 4, \dots$. These are polynomial systems of the form:

$$\begin{cases} c_1 x^{a_1} y^{b_1} + c_2 x^{a_2} y^{b_2} + c_3 x^{a_3} y^{b_3} = 0 \\ \sum_{i=1}^m c'_i x^{a'_i} y^{b'_i} = 0 \end{cases}$$

For these systems, Khovanski [Kho91] obtained the bound

$$\mathcal{N}'(3, m) \leq 3^{m+2} 2^{(m+2)(m+1)/2}$$

This bound, for $m = 3$, gives 24882, a number which is considerable larger than the actual value. In [LRW03], the authors show a much better bound, together with the equality between the maximum number of isolated and non-degenerate roots:

$$\mathcal{N}'(3, m) = \mathcal{N}(3, m) \leq 2^m - 2.$$

Moreover, they show that $\mathcal{N}'(3, 3) = \mathcal{N}(3, 3) = 5$. This means that the bound $2^m - 2$, although a lot better than Khovanski’s one, is not optimal, since for $m = 3$ it evaluates to 6. For $m = 4$, the previous bound gives 14. The main motivation for the experiments of this work were to find numerical evidence that $\mathcal{N}(3, 4)$ is strictly less than 14.

The main trick on [LRW03] consists in reducing the fewnomial system F to a univariate polynomial, with a very particular structure, and bound its maximum number of roots in an interval via elementary Rolle’s theorem applications. We formulate the reduction for our special case here, since we will also use it.

Lemma. *Let $F = (f_1, f_2)$ be a 2×2 fewnomial system of type $(3, m)$. Assume that $\text{Newt}(f_1)$ has dimension 2, that is, there are two non-zero vector exponents linearly independent. Then, there exist (a_1, \dots, a_{m-1}) , (b_1, \dots, b_{m-1}) and (c_1, \dots, c_{m-1}) in \mathbb{R}^n such that F has the same number of non-degenerated non-isolated positive roots as*

$$f(x) = 1 + c_1 x^{a_1} (1-x)^{b_1} + \dots + c_{m-1} x^{a_{m-1}} (1-x)^{b_{m-1}}.$$

Proof. We can assume, without loss of generality, that f_1 and f_2 have an independent term 1, dividing by a suitable monomial. Take a_1 and a_2 the two linearly independent vector exponents of f_1 and build the matrix A which has those vectors as columns. By construction we have that A is invertible. With the substitution $x \mapsto x^{A^{-1}}$, we send $f_1 \mapsto 1 + c_1 x + c_2 y$, and this map preserves the conditions on isolation and degeneracy.

Moreover, in order for F to have some positive root, it cannot happen that all the coefficients of f_1 to be positive. Then, choosing the term with different sign from the others in the first normalization and then applying the mapping $(x, y) \mapsto (x/|c_1|, y/|c_2|)$, we obtain $f_1 \mapsto 1 - x - y$. Substituting y in f_2 we get the univariate polynomial of the thesis. \square

We show an application of this Lemma below. It is clear that it is enough to look for roots of the new univariate polynomial in the interval $(0, 1)$, since the equation $1 - x - y = 0$ implies $x, y \leq 1$ in the positive orthant. In what follows we will call *univariate fewnomial* or *univariate representation* to this kind of polynomial, for easy reference.

A. Kouchnirenko proposed a bound for a system of fewnomial of this kind thirty years ago. Kouchnirenko's conjecture was that $\mathcal{N}(m_1, \dots, m_k) \leq (m_1 - 1) \cdots (m_k - 1)$. Bertrand Haas [Haa02] shows a counterexample for this conjecture:

$$\begin{aligned} y^{108} + 1.1x^{54} - 1.1x &= 0, \\ x^{108} + 1.1y^{54} - 1.1y &= 0. \end{aligned} \tag{1}$$

This system has five positive roots, while Kouchnirenko's conjecture predicts less than $(3 - 1) \times (3 - 1) = 4$ roots. In the next section we show two verifications of this example.

The Haas' example is constructed from a system of type $(3, 3)$ with three positive roots and then making a diffeomorphic deformation for adding two more roots. This kind of deformation strongly uses the symmetry of a $(3, 3)$ system, thus, it seems hard to generalize to a system of type $(3, 4)$. Using the winding number calculator we could find fewnomials of type $(3, m)$, for some $m \geq 3$, with 7 positive roots.

We apply the lemma to this example for constructing the univariate fewnomial, which we will use in the next section. Dividing the first polynomial by $-1.1x$ and the second by $-1.1y$ we get the equivalent system:

$$\begin{aligned} -(1/1.1)x^{-1}y^{108} - x^{53} + 1 &= 0 \\ -(1/1.1)x^{108}y^{-1} - y^{53} + 1 &= 0. \end{aligned}$$

Then, we define $A = \begin{pmatrix} -1 & 53 \\ 108 & 0 \end{pmatrix}$, which is clear invertible, and we apply the transformations stated in the proof of the previous lemma. That gives the univariate fewnomial

$$1 + 1.1^{-109/108}x^{-1/108}(1-x)^{11663/5724} - 1.1^{53/108}x^{53/108}(1-x)^{1/108} = 0. \tag{2}$$

We briefly return to the bound $2^m - 2$ for $\mathcal{N}(3, m)$. The technique for bounding the positive roots of the univariate fewnomial, is to derive and to divide by suitable terms of the form $x^\alpha(1-x)^\beta$, which neither introduce nor remove roots. Doing the operations in the correct order one can reduce the number of term of the univariate fewnomial, until obtaining a expression of the form $x^\alpha(1-x)^\beta p(x)$, where p is a polynomial (see [LRW03] or the solution of the exercise 2 of homework 2). The final step is to bound the number of

roots of p by its degree and take into account the number of derivatives. It is reasonable, then, to ask whether, some better bound can be use to this particular polynomial, perhaps taking into account the successive coefficients of p and using the Descarte's rule.

The following example show that it is not possible to improve this process. Consider the univariate fewnomial of type (3, 4)

$$1 + (1 - x)^2 - (1 - x)^3 + x^4(1 - x)^4. \quad (3)$$

Running the previous process over this polynomials yield

$$1152 - 69312x + 946560x^2 - 5172480x^3 + 13796160x^4 \\ - 19130496x^5 + 13257216x^6 - 3628800x^7,$$

so, in this case the Descarte's bound coincides with the degree, giving 14 after adding the number of derivatives. The univariate fewnomial, however, has 0 positive roots, as can be verified with the winding number calculator (see 4).

2 Experiments and results

2.1 Existing polynomial solvers

The first step to trying to count the number of roots of a fewnomial system was to feed them to some of the existing solvers. *Mathematica* has solvers of systems of equations, which are fast for polynomials but they are a lot slower for our extended polynomial with real exponents. Moreover, even the in fastest case, they are not useful for computing solutions of large samples. Also, the solver is not completely reliable. For example:

```
In[67] := g
Out[67] = -6.5442500559621015*(1 - x)^6*x^2
          + 7.405506950749242*(1 - x)^7*x^5
          - 7.669308187403162*(1 - x)^7*x^9
          + 4.778114064117451*(1 - x)^6*x^10

In[68] := Select[x /. NSolve[g==0, x],
                (Im[#]==0 && 0<Re[#]<1)&]
Out[68] = {0.989684,0.998774,0.998774}
```

The solver reports three roots near one, but actually this polynomial has 0 roots in $(0, 1)$.

The other polynomial solver tried was `phc`, from Jan Verschelde [Ver99]. This application implements a lot of different algorithms for homotopy continuation. It is oriented as an application for finding all the roots with many adjustable parameters. It can also compute mixed volumes and calculates initial systems for the homotopies based in several kind of methods. However, since we are

interested in just counting roots, this application is overkill, together with the fact that it takes too much time.

We run the Haas' example in `phc`, with the default parameters and we obtained the summary information:

```
=====
A list of 11664 solutions has been refined :
Number of regular solutions   : 11664.
Number of singular solutions  : 0.
Number of real solutions      : 6.
Number of complex solutions   : 11658.
Number of clustered solutions : 0.
Number of failures            : 0.
=====
```

...*(snipped information)*...

The total elapsed time is 956 seconds = 15 minutes 56 seconds.

`phc` reported six real roots, one of them is 0, which gives the correct count. This can be verified searching in the list of the 11664 solutions that it outputs. Observe that it took more than fifteen minutes. This makes this application unusable for running many examples. Another problem with this program is that the parameters need to be adjusted in different ways, depending on the polynomial. This is impossible to do when selecting random coefficients or exponents.

For example, the perturbation of the Haas' example

$$\begin{aligned} y^{108} + 1.1x^{54} - 1.1x + x^{108}y^{54} &= 0, \\ x^{108} + 1.1y^{54} - 1.1y &= 0. \end{aligned}$$

took 1 hour and 51 minutes and it reported

```
=====
A list of 17496 solutions has been refined :
Number of regular solutions   : 14580.
Number of singular solutions  : 0.
Number of real solutions      : 4.
Number of complex solutions   : 14576.
Number of clustered solutions : 0.
Number of failures            : 2916.
=====
```

Observe the big number of failures in the paths. If we also inspect the real roots found by `phc`, we found that one root is zero, and the other three are very near to each other and near to 1. The winding number calculator reports only one positive root for this example.

This tests show that a faster and more reliable ways to count the real roots is needed if we want to operate with large sets of fewnomials to estimate its maximum value.

2.2 Winding numbers

This idea is from Prof. Rojas and trying to implement it is the core of this work. The basic trick consists in using the principle of the argument to count the number of complex roots of an analytic function. We state this principle for completeness.

Theorem. *Let $f : \Omega \rightarrow \mathbb{C}$ be an analytic function and assume that $\partial\Omega$ is a path in \mathbb{C} which does not pass through any zero of f . Then, the number of zeros of f in the interior of Ω is the winding number of $f(\partial\Omega)$ around 0:*

$$\frac{1}{2\pi i} \int_{f(\partial\Omega)} \frac{1}{z} dz.$$

Then, given an integer $k > 2$, we define the curve \mathcal{C} as in the following figure. If we take k big enough and we count the number of roots inside it

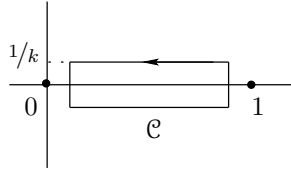


Figure 1: the region of integration

with the previous theorem, we will be counting the roots of f in the interval $(0, 1)$. Therefore, the main problem is to implement an accurate and fast way to compute the integral.

First observe that the integral can be written

$$\frac{1}{2\pi i} \int_{f(\mathcal{C})} \frac{1}{z} dz = \frac{1}{2\pi} \Delta_{\mathcal{C}} \text{Arg } f(z),$$

where $\Delta_{\mathcal{C}} \text{Arg } z$ is the difference from the beginning to the end of \mathcal{C} of any continuous argument of z over \mathcal{C} .

Given $z_1, z_2 \in \mathbb{C}$, we denote by $[z_1, z_2]$ the complex segment $\{z_1 + t(z_2 - z_1) \mid t \in [0, 1]\}$. Assume, as it is our case, that \mathcal{C} is a polygonal path. Then, when z_1 and z_2 are two points of \mathcal{C} , with $[z_1, z_2] \subset \mathcal{C}$, close enough such that $|\Delta_{[z_1, z_2]} \text{Arg } f(z)| \leq \pi$, then the variation on the argument can be calculated by

$$\Delta_{[z_1, z_2]} \text{Arg } f(z) = \text{Arg}(f(z_2)/f(z_1)),$$

where Arg is the standard argument function with range in $(-\pi, \pi]$. This is clear, since under the previous hypothesis, the function Arg is continuous. In summary, we can compute the winding number as

$$\int_{f(\mathcal{C})} \frac{1}{z} dz = \frac{1}{2\pi} \sum_{i=1}^N \text{Arg}(f(z_{i+1})/f(z_i)),$$

with $z_{N+1} = z_1$, provided that $|\Delta_{[z_i, z_{i+1}]} \text{Arg} f(z)| \leq \pi$ for all i .

It is important to remark that satisfying this condition is the main work in the implementation of the calculator. A first and naïve approximation is to consider a big number N and equally distributed points z_i over \mathcal{C} . Since f is continuous, when N goes to $+\infty$, the previous sum converges to the correct value.

A very simple implementation of this technique in *Mathematica* is shown here

```

line[x_, y_][t_] := (y - x)t + x

wind[F_, k_, N_] := Module[
  {f, v},
  f[x_ /; 0 <= x < 1/4] := line[1 - 1/k - I/k,
                                1 - 1/k + I/k][4x];
  f[x_ /; 1/4 <= x < 1/2] := line[1 - 1/k + I/k,
                                1/k + I/k][4x - 1];
  f[x_ /; 1/2 <= x < 3/4] := line[1/k + I/k,
                                1/k - I/k][4x - 2];
  f[x_ /; 3/4 <= x <= 1] := line[1/k - I/k,
                                1 - 1/k - I/k][4x - 3];
  v = Composition[F, f] /@ Range[0, 1, 1/N];
  (Plus @@ Most[MapThread[Arg[#1/#2]&,
                        {RotateLeft[v], v}]])/(2Pi)
]

```

The parameters of `wind` are the function, the closeness of the curve to the axis and the number of points, which are equally distributed on each side of the rectangle \mathcal{C} . The function `wind` can successfully count five roots on the Haas' example for $N = 5000$ in little more than 1 minute. However, in the same way as with `phc`, other examples require N of an order of magnitude bigger. Some examples of fewnomials of type $(3, 3)$ reported more than 6 roots with this method, which is clearly wrong. The problem, as before, is that the number and position of the points z_i is highly dependent on the function F .

2.3 A reliable winding number algorithm

Ying and Katz [YK88] provide an extremely fast and reliable algorithm for computing the winding number of an analytical function. We describe here the

main theorems and how they are applied to our case of the univariate fewnomial. Most of this section is extracted from [YK88] and [Hen74].

As we said before, the main problem is to decide when the variation of the argument of f in a complex segment is less than π . The following theorem gives a sufficient condition.

Theorem. *Let f be a complex function with second derivative on the interval $[z_1, z_2]$. Define*

$$P(z) = f(z_1) + (z - z_1) \frac{f(z_2) - f(z_1)}{z_2 - z_1},$$

$$R(z) = 1/2M(z - z_1)(z - z_2),$$

where M is a bound for the modulus of the second derivative of f in $[z_1, z_2]$. Then, if $|R(z)| < |P(z)|$ for all $z \in [z_1, z_2]$, then $|\Delta_{[z_1, z_2]} \text{Arg } f(z)| \leq \pi$.

The proof of this theorem and all the others lemmas and results from [YK88] are based just in standard techniques of complex analysis. However, the results are fairly useful.

To use the previous theorem, one must be able to decide when $|R(z)| < |P(z)|$ in the interval $[z_1, z_2]$. Substituting in values $z - z_1$ and $z - z_2$ by $t(z_2 - z_1)$ and $(1 - t)(z_2 - z_1)$, respectively, we have the condition

$$|f(z_1) + t(f(z_2) - f(z_1))| > G(z_1, z_2)t(1 - t),$$

for all $t \in [0, 1]$, where $G(z_1, z_2) = M(z_1, z_2)|z_2 - z_1|^2/2$. The following theorem gives a computable way to test this condition in t .

Theorem. *Let $a, b \in \mathbb{C}$ not null and $G \geq 0$. Assume that $m = \min|a + t(b - a)| \neq 0$ in $[0, 1]$. Let $H = G + |b - a|^2/(2m)$. Then, if $|a + t(b - a)| \leq Gt(1 - t)$ for some $t \in [0, 1]$, then $t^* = 1/2 - (|b| - |a|)/(2H)$ verifies such inequality and $t^{*2} \geq |a|/H$.*

Therefore, it is enough to compute t^* to decide that the condition is the first theorem is true. The only value still to be calculated is the minimum in $t \in [0, 1]$ of the linear function $a + t(b - a)$. The following simple lemma computes this minimum.

Lemma. *Let $a, b \in \mathbb{C}$ and let $t^* \in [0, 1]$ is the value that attains the minimum of $|a + t(b - a)|$ in $[0, 1]$. Then,*

- *If $\text{Re } a\bar{b} \geq |a|^2$, then $t^* = 0$.*
- *If $\text{Re } a\bar{b} \geq |b|^2$, then $t^* = 1$.*
- *Otherwise, $t^* = (|a|^2 - \text{Re } a\bar{b})/|b - a|^2$.*

With this ingredients, we can decide very fast if the variation of the argument of f in $[z_1, z_2]$ is less than π . When the condition of the second theorem is not

satisfied, we have an intermediate point, $z^* = z_1 + t^*(z_2 - z_1)$, and we try again with the intervals $[z_1, z^*]$ and $[z^*, z_2]$. This is the heart of the algorithm.

The function f , in our case, is the univariate fewnomial, which is analytic in the region enclosed by \mathcal{C} . Therefore, it has second derivative, which can be calculated algebraically. In our implementation we included the form of the second derivative of a univariate fewnomial, and we calculate its maximum evaluating this function on a grid of $[z_1, z_2]$. This is the most time consuming individual step, and can be improved plugging in a better algorithm in the implementation (see 2.5).

Once we know that the argument of f varies less than π over each of the $[z_i, z_{i+1}]$, we can compute it with usual formula. However, the calculation of $\text{Arg}(f(z_2)/f(z_1))$ requires the calculation of an inverse trigonometric function, which has high cost. Henrici [Hen74] proposes a very fast way to calculate it without using trigonometric functions. It consists on dividing the complex plane in eight sectors as show in the figure: Then, the variation of the argument from

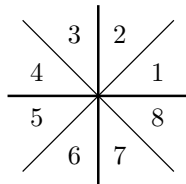


Figure 2: Henrici’s algorithm

$f(z_1)$ to $f(z_2)$ is the number of sectors between these two points, divided by eight. If the number of sectors is four, then we cannot decide the sign of the variation. In that case, we assume that the test for variation less than π failed, and we find an intermediate point to repeat the process. In all the other cases it is obvious how to decide the sign of the increment.

Observe that to decide in which sector lie $f(z)$, it is enough to do comparisons between the real and the imaginary parts of $f(z)$, and no other operation. In the implementation, the tests with this method and with the usual inverse of the arctan yielded similar results in time, probably because of the very good implementation of this function in the GNU Scientific Library (see 3). Although the Henrici’s method gives more “precise” integers, and can be useful when the cost of evaluating the function f with enough precision is high. In our case, the evaluation of the fewnomial is not too high.

We give more details on the implementation, together with a documentation of the features, in the next section.

2.4 Results of the experiments

We implemented the previous algorithm in C, building an application called `wind`. We also wrote a prototype in *Mathematica*, which was too slow to be of

any use. The program `wind` is an option driven application, in which the data of the fewnomials is passed through a file. The program and the code is free to be downloaded and modified (see 3.1).

We show briefly the options of the program and, then, we show the results of some runs. When invoked without parameters it displays

```
Missing file (-f)
wind <options>
Compute the number of positive roots of fewnomial systems

-v          print information during calculations
-f file     read from 'file'
-t          'file' is a fewnomial system (default
           univariate polynomial)
-k number   size of the integration curve (default 10000)
-n number   number of iterations for exponents (default 1)
-m number   number of iterations for coefficients for
           each exponent (default 1)
-l          list the possible generators of random numbers
-r name     use the random generator 'name' (default 'taus')
-s seed     seed for the generator (default 0)
```

See docs for more details

The option `-f` indicates the file in which the data of the system is included. One can feed `wind` with either the fewnomial system directly or the univariate representation of the system. In any of both case, the first line of the file must have a line of the form

(a, b)

which is the dimension of the matrix of exponents and coefficients. When feeding the fewnomial system, we must write a matrix whose first row are the exponents of the ' x ' of the polynomials, the second row are the exponents of ' y ' and the third rows are the coefficients. Thus, in the case of a fewnomial system of type (n, m) , we have $a = 3$ and $b = n + m$. For example, the Haas' system can be expressed as

```
(3, 6)
0
54
1
108
0
0
108
0
0
```

```

0
54
1
1
1.1
-1.1
1
1.1
-1.1

```

Any of the entries of the matrix can be substituted by an expression of the form, for example,

```
* -2 3 u   or   * 2 0.1 g
```

The first expression denotes a random entry with uniform distribution in the interval $(-2, 3)$. The second one denotes a Gaussian entry with mean 2 and variance 0.1. With this feature it is possible to generate different kinds of random fewnomials.

The other form of entering the data of the fewnomial is with its univariate representation. In this case, the first row of the matrix are the exponents of ' x ', the second row are the exponents of ' $(1 - x)$ ' and the third row are the coefficients of the univariate polynomial. Random entries can also be used. The option `-t` signal which format we are using.

The option `-k` denote how close is \mathcal{C} to the real axis (see figure 2.2). The option `-n` denotes how many random samples for the exponents must be extracted. For each realization of the exponents, the option `-m` controls the number of samples for the coefficients. This is to allow that the fewnomials reach many points outside of a fixed discriminant variety.

The options `-r` and `-s` tune up the random generators. Since we are generating many random numbers and the fewnomial systems are very sensible to them, it seems reasonable to have the possibility to test different generators. In large samples it may be, also, considerable difference in speed between them.

It is important to remark that given a generator and a seed, the numbers generated are deterministic; thus, any of the following examples can be verified on any computer. The files for the examples are included in the source distribution (see 3.1).

A run of `wind` with the file shown above, that is, the Haas' example, gives:

```

$ ./wind -f haas.dat -v -t
(3, 3) fewnomial system
Integration along a 0.9998 x 0.0002 rectangle
1 iteration(s) for exponents
1 iteration(s) for coefficients for each exponent
(total number of iterations: 1)
Using random generator 'taus' with seed 0

```

```

-----
1 x^(0) y^(108) + 1.1 x^(54) y^(0) + -1.1 x^(1) y^(0) = 0
1 x^(108) y^(0) + 1.1 x^(0) y^(54) + -1.1 x^(0) y^(1) = 0
real roots in (0,1): 5 (float value 5)
numbers of sampled points: 68
CPU elapsed time: 0.2 secs

```

```

-----
Maximum number of roots in (0,1): 5
Total number of fewnomial systems: 1 (out of 1)
Average number of sample points: 68
Total (CPU) elapsed time: 0.2 secs

```

Observe that the number of points needed around \mathcal{C} for computing the winding number were 68, against the 5000 needed in the naïve implementation. Also, the time spent was just 0.2 seconds.

We can use `wind` to check the number of roots of the polynomial (3):

$$1 + (1 - x)^2 - (1 - x)^3 + x^4(1 - x)^4. \quad (4)$$

Feeding the program with the (3,3) matrix of its coefficient and exponents, it produces the output

```

$ ./wind -f descar.dat -v
...{snipped output}...
Maximum number of roots in (0,1): 0
Total number of fewnomial systems: 1 (out of 1)
Average number of sample points: 3
Total (CPU) elapsed time: 0 secs

```

The number of sample points is the number of additional points that need to be calculated. So, the total number of z_i used is 7.

The next experiment was to do some perturbations to this example. It turned out that simulating the fewnomial systems is more unstable than simulating the univariate representation. Many samples produce a matrix A too near the singular variety. We choose in that case to just discard the system and to continue, instead of aborting. In some other cases, the number of points to compute the winding number increases out of stack. We also choose to ignore and discard this kind of systems.

Therefore, we use the univariate representation of the Haas' example calculated in (2):

```

(3,3)
-0.0092592592592592587
0.49074074074074076
* -1 1 u
2.0375611460517122
0.0092592592592592587

```

```

* -1 1 u
-0.908288988750195
-1.0478836809063943
* -2 1 u

```

adding another uniform random term for constructing a (3, 4) fewnomial. The run gives

```

$ ./wind -f haas34.dat -n 10 -m 100 -s 18 -k 1000000 -r ranlux
(3, 4) fewnomial univariate representation
Integration along a 0.999998 x 2e-06 rectangle
10 iteration(s) for exponents
100 iteration(s) for coefficients for each exponent
(total number of iterations: 1000)
Using random generator 'ranlux' with seed 18

Estimated time for calculation: 195 secs
-----
Maximum number of roots in (0,1): 5
Total number of fewnomial systems: 1000 (out of 1000)
Average number of sample points: 46
Total (CPU) elapsed time: 231.41 secs

```

Observe that we simulate 10 exponents and 100 coefficients for each exponent set. The program also reported 5 positive roots.

The generation of Gaussian samples is useful to make small perturbations. For example, perturbing the coefficients of the fewnomial system (not the univariate representation) by a normal random variable with mean in the corresponding exponent or coefficient and with variance 0.1:

```

./wind -f haasgauss.dat -t -n 1000 -k 10000 -s 19

```

also yields 5 roots for some fewnomials of type (3, 3). Thus, we have constructed new examples of systems which attains the maximum $\mathcal{N}(3, 3)$.

The next step was try to find examples of fewnomials of type (3, 4) with more than 5 roots, which were not modifications of the Haas' system. We then set a fewnomial system with random uniform exponents in the interval (0, 10) and coefficients uniform in the interval (-1000, 1000). The following runs gave 5 and 6 positive roots, respectively:

```

./wind -f few34unif.dat -t -n 1000 -k 10000 -s 23 -r ranlux
./wind -f few34unif.dat -t -n 1000 -k 1000000 -s 34
-r knuthran

```

The following system, according to `wind`, has 6 positive roots:

$$\begin{aligned}
 &181.753x^{2.79717}y^{7.504} - 258.419x^{4.43991}y^{5.39554} \\
 &\qquad\qquad\qquad - 114.636x^{2.25739}y^{8.30766} = 0 \\
 &256.763x^{7.11364}y^{0.255148} + 882.215x^{2.36632}y^{7.71484} \\
 &\qquad\qquad\qquad - 839.223x^{0.0107616}y^{1.89108} + 531.323x^{2.44404}y^{8.08467} = 0
 \end{aligned}$$

Some runs with the `few34unif` gave 7 roots. But after inspecting the generated system, it turned out that all the coefficients of the first polynomial had the same sign, which makes that the fewnomial system do not verify the hypothesis of the reduction lemma. This must be checked and fixed in improved versions of `wind`.

We test, also, fewnomials of type (3, 5) and (3, 6). The fewnomial of type (3, 5) gave 4 roots:

```
./wind -f few35.dat -t -k 100000 -n 1000 -s 10 -r knuthran
```

The fewnomial of type (3, 6) yielded the amazing result of 11 positive roots. However, inspecting the generated system visually, it has two exponents vectors of the first polynomial very near. This makes its Newton polytope to be of dimension 1, and we are not in the hypothesis to reduce it to an univariate case. Again, this kind of check must be included in the program. All the others samples of `few36` produce less than 5 roots:

```
./wind -f few6.dat -t -n 1000 -k 1000000 -s 51 -r knuthran
```

The known bound predicts less than $2^5 - 2 = 30$.

The case of a fewnomial of type (3, 8) yielded 12 positive roots. There was two systems with 12 positive roots, both of them in the hypothesis of the reduction lemma. And there was also a system with 10 positive roots, in the same hypothesis.

```
./wind -f few39.dat -t -n 1000 -k 1000000 -s 10 -r knuthran
```

In this case, the known bound is $2^8 - 2 = 254$.

2.5 Conclusions and further developments

According to the previous experiments, one can conjecture that the increase on the number of positive roots of fewnomials of type (3, m) is much less than exponential, as it is the best bound known: $2^m - 2$. From all the experiments, only one systems of type (3, 4) appears to have more than 5 roots, actually 6. And three systems of type (3, 8) surpass the number 5, having 10, 12 and 12 number of positive roots, respectively. Although the program seems very reliable, this examples should be checked by other means, to ensure the number of positive roots. Trying to solve the (3, 4) fewnomial with *Mathematica* was unsuccessful. Also, the program `phc` does not work with real exponents, so we cannot use it for verification.

Without those examples, one may be tempted to propose 5 as a bound for $\mathcal{N}(3, m)$, independent of m . We have been unable to find other examples, with more than five roots. That's the importance of verifying them. It is at least strange that the results are more concentrated below 5, regardless of m . This can be checked invoking `wind` with the option `-v`, which displays the individual systems generated and their number of roots.

But the main conclusion is that a lot more work is needed in determining better ways of sampling the fewnomial systems, in order to reach all the connected regions of the complement of the discriminant variety. Randomizing the univariate representation seems to be less efficient than randomizing the systems. We suppose this is due to the fact that varying the $m - 1$ coefficients of the univariate form does not suffice to make the polynomial reach the different components. Randomizing the coefficients of the fewnomial system, we are sure to cover the complement of the multidimensional discriminant in a more uniform way. This gives strength to the other idea of Prof. Rojas, not explored here, about trying to find a multidimensional approach, without using the univariate reduction.

With respect to the application, it also requires a lot of work to make it more robust with respect to errors and exceptions. In the actual state, it does not verify neither the input files nor the parameters supplied. Some sections can be improved, for example, the computation of the maximum, which is done in a very naïve way.

It also needs to check whether the random generated systems verify the condition allowing them to be reduced to an univariate polynomial, before trying to compute the number of roots.

3 Implementation

3.1 The program

The core of this program is a mostly straightforward implementation of the Ying and Katz [YK88] and Henrici [Hen74] algorithms. It is focused in fewnomial systems of type $(3, m)$, since it is for this kind of system that we can do the reduction to the univariate form.

The code can be freely used, modified. It is available in

`http://www.math.tamu.edu/~wmoreira/wind`

in source and statically compiled form. The source package includes the examples mentioned in the previous section. The statically compiled binary is for Linux. If you want to compile the sources, just make sure to have installed the *GNU Scientific Library* (www.gnu.org/software/gsl), and execute `make`.

This program is written in standard C, and relies in the GNU Scientific Library, for operating with complex numbers and random generators.

There are some variables which can be tweaked in compilation time. If you are getting many discarded systems (it can be seen in the summary that `wind`

produces at the end), you can try to enlarge the size of the stack (`MAX_STACK`) or to reduce the maximum and minimum coefficients allowed (`MIN_COEFF` and `MAX_COEFF`). Also, the number of points for computing the maximum of the second derivative (`MAX_GRID`) and some different precisions limits can be modified.

3.2 The code

In this section we give some details on the implementation of the algorithm.

The three main functions are `less_pi`, `wind` and `process`. The function `less_pi` returns true when it can decide that the variation of the argument is less than π . When it returns false, it modifies the parameter `t` to an intermediate point, as given by the theorem 2.3:

```
int
less_pi(gsl_matrix *f, point z1, point z2, double *t)
```

Its implementation is almost verbatim from [YK88]. In all this functions, the parameter of type `point` includes the complex point z and its functional value $f(z)$. This is so for minimizing the number of evaluations of the fewnomial univariate function.

The function `wind` has signature

```
double
wind(gsl_matrix *f, gsl_complex z1, gsl_complex z2,
     int *numpoints, int *err)
```

It computes the variation of the argument over the segment $[z_1, z_2]$ of f , invoking `less_pi` and the Henrici's algorithm (see 2.3) when the variation is less than π . This implementation varies from the one proposed by Ying and Katz, since it is iterative, but the efficiency is the same. This implementation tries to avoid the overhead of recursion by setting its own very specific stack.

There are two ways in which this function can fail. One is because the stack of points being calculated gets full. This usually happens when `less_pi` fails too often and too many subdivisions of a segment must be made. It means that f has big coefficients or exponents and $f(\mathcal{C})$ gets too convoluted.

The other reason in which `wind` fails is when there is zero very near the integration path. In these cases, a function `detour` is called to try calculate a path around the zero. This function is not implemented in this application and is a good improvement for further versions. In this implementation, `detour` just signal an error in the variable `err`.

The parameter `numpoints` is reseted to the number of points that were needed to satisfy the "less than π " condition.

The function `process` is the main loop of the application. It loads the data from the file and set two loops according to the number of iterations for the exponents and coefficients. In each iteration it generates the random numbers with the respective distributions and makes some checks about the results. For example, in the case of an univariate fewnomial, it verifies that not all coefficients

are positive (otherwise it is trivial) and checks the sizes of them. As noted in the *Conclusions* section, more checks are needed at this point.

The last significant function is

```
gsl_matrix *  
few_to_poly(gsl_matrix *g)
```

This function computes the univariate representation of a $(3, m)$ fewnomial. It returns NULL when the matrix A from lemma 1.1 is too near the singular variety. It also limits the size of the resulting coefficients. Better ways to choose the monomials for normalization can greatly improve the stability of the resulting univariate polynomial.

References

- [Haa02] Bertrand Haas. A simple counterexample to Kouchnirenko’s conjecture. *Beiträge Algebra Geom.*, 43(1):1–8, 2002.
- [Hen74] Peter Henrici. *Applied and computational complex analysis*. Wiley-Interscience [John Wiley & Sons], New York, 1974. Volume 1: Power series—integration—conformal mapping—location of zeros, Pure and Applied Mathematics.
- [Kho91] A. G. Khovanskiĭ. *Fewnomials*, volume 88 of *Translations of Mathematical Monographs*. American Mathematical Society, Providence, RI, 1991. Translated from the Russian by Smilka Zdravkovska.
- [LRW03] Tien-Yien Li, J. Maurice Rojas, and Xiaoshen Wang. Counting real connected components of trinomial curve intersections and m -nomial hypersurfaces. *Discrete Comput. Geom.*, 30(3):379–414, 2003.
- [Ver99] Jan Verschelde. Pchpack: A general-purpose solver for polynomial systems by homotopy continuation. *ACM Transactions on Mathematical Software*, 25(2):251–276, 1999.
- [YK88] Xingren Ying and I. Norman Katz. A reliable argument principle algorithm to find the number of zeros of an analytic function in a bounded domain. *Numer. Math.*, 53(1-2):143–163, 1988.